Formal verification of Mobile Robot Protocols

Béatrice Berard, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, Sébastien Tixeuil

Formal Verification of Mobile Robot Protocols

Béatrice Bérard, Laure Millet, Maria Potop-Butucaru,
Yann Thierry-Mieg and Sébastien Tixeuil.
Université P. et M. Curie, LIP6, Paris, France
E-mail: *firstname.lastname@lip6.fr*

June 14, 2013

Abstract

Mobile robot networks emerged in the past few years as a promising distributed computing model. Existing work in the literature typically ensures the correctness of mobile robot protocols via *ad hoc* handwritten proofs, which, in the case of asynchronous execution models, are both cumbersome and error-prone.

In this paper, we propose the first formal model and general verification (by model-checking) methodology for mobile robot protocols operating in a discrete space (that is, the set of possible robot positions is finite). Our contribution is threefold. First, we formally model using synchronized automata a network of mobile robots operating under various synchrony (or asynchrony) assumptions. Then, we use this formal model as input model for the DiVinE model-checker and prove the equivalence of the two models. Third, we verify using DiVinE two known protocols for variants of the ring exploration in an asynchronous setting (exploration with stop and perpetual exclusive exploration).

The exploration with stop we verify was manually proved correct only when the number of robots is $k > 17$, and n (the ring size) and k are co-prime. As the necessity of this bound was not proved in the original paper, our methodology demonstrates that for several instances of k and n *not covered* in the original paper, the algorithm remains correct. In the case of the perpetual exclusive exploration protocol, our methodology exhibits a counter-example in the completely asynchronous setting where safety is violated, which is used to correct the original protocol.

1 Introduction

The variety of tasks that can be performed by autonomous robots and their complexity are both increasing [1]. Many applications envision groups of mobile robots self-organizing and cooperating toward the resolution of common objectives, in the absence of any central coordinating authority.

A recent trend was to shift from the classical *continuous* setting where robots evolve in a continuous two-dimensional Euclidian space, to a discrete one where space is partitioned into a *finite* number of locations. The discretization process is motivated by practical aspects with respect to the unreliability of sensing devices used by the robots as well as inaccuracy of their motorization [2]. This discrete space is conveniently represented by a graph, where nodes represent locations, and edges represent the possibility for a robot to move from one location to the other. While the discrete setting permits to simplify robot models by reasoning on finite structures, it significantly increases the number of symmetric configurations when the underlying graph is also symmetric (*e.g.* a ring) and thus the complexity of proving the correctness of protocols [3, 4, 5].

We consider a distributed system of k mobile robots that have limited capabilities: they are identical and anonymous (they execute the same algorithm and they cannot be distinguished using their appearance), they are oblivious (they have no memory of their past actions) and they have neither a common sense of direction, nor a common handedness (chirality). Furthermore robots do not communicate in an explicit way. However they have the ability to sense the environment and see the position of the other robots. Robots operate in three phase cycles: *Look*, *Compute* and *Move*. During the *Look* phase robots take a snapshot of the graph together with other robots' positions. The collected information is used in the *Compute* phase in which robots decide to move or to stay idle. In the *Move* phase, robots may move to one of their adjacent nodes computed in the previous phase. In the original model introduced by Suzuki & Yamashita [6] (that described two variants [1]: FSYNC, for fully synchronous, and SSYNC, for semi-synchronous), an arbitrary non-empty subset of robots execute the three phases synchronously and atomically. This model was later generalized by Flocchini *et al.* [7] to handle full asynchrony and remove atomicity contraints (this model is called ASYNC [1], for asynchronous, in the sequel). One of the key differences between the FSYNC, SSYNC, and ASYNC models in the discrete setting is that the ASYNC model allows a robot to compute and move based on an *outdated* view of the system. It is notorious that handwritten proofs for protocols operating in the ASYNC model are hard to write and read, due to many instances of case-based reasoning that is both cumbersome and error-prone.

Model-checking [8, 9] is an appealing tool for verifying safety and liveness properties of finite systems, and has been successfully used for the verification of various distributed systems [10, 11, 12, 13, 14, 15, 16, 17] ranging from classical shared memory (consensus, transactional memory) to population protocols. To our knowledge, in the context of mobile robots operating in discrete space, only two previous attempts, by Devismes *et al.* [18] and by Bonnet *et al.* [19], investigate the possibility of automating verification of mobile robots protocols. The first paper uses LUSTRE to describe and model-check the problem of exploration with stop of a 3 grid by 3 robots in the SSYNC model, and to show by exhaustive searching that no such protocol can exist. The second paper considers the perpetual exclusive exploration by k robots of n-sized rings, and mechanically

generates all *unambiguous* protocols for k and n in the SSYNC model (that is, all protocols that do *not* have symmetric configurations). Those two works differ from our proposal in several ways. First, they are restricted to the simpler SSYNC model rather than the more general and more complex ASYNC model. Second, they are either specific to a hardcoded topology (*e.g.*, a 3 grid [18]) that prevents easy reuse in more generic situations, or make additional assumptions about configurations and protocols to be verified (*e.g.* unambiguous protocols [19]) that prevent combinatorial explosion but forbid reuse for proof-challenging protocols, which would most benefit from automatic verification.

In this paper, we propose the first formal model and general verification (by model-checking) methodology for mobile robot protocols operating in a discrete space. Our contribution is threefold. First, we formally model using synchronized automata a network of mobile robots operating under various synchrony (or asynchrony) assumptions (namely, FSYNC, SSYNC, and ASYNC). We use linear temporal logic (*a.k.a.* LTL in the sequel) to specify the mobile robots tasks and permit good expressivity and versatility. Then, we use this formal model as input model for the DiVinE model-checker and prove the equivalence of the two models. Third, we verify using DiVinE two known protocols for variants of the ring exploration in an asynchronous setting (exploration with stop [20] and perpetual exclusive exploration [21]).

The exploration with stop protocol [20] we verify was manually proved correct only when the number of robots is $k > 17$, and n (the ring size) and k are co-prime. As the necessity of this bound was not proved in the original paper, our methodology demonstrates that for several instances of k and n *not covered* in the original paper, the algorithm remains correct. In the case of the perpetual exclusive exploration protocol [21], our methodology exhibits a counter-example in the completely asynchronous setting where safety is violated, which is used to correct the original protocol and we later verify the correction.

2 Model-checking background

We first recall the definitions of finite automata, synchronized products and LTL specifications.

Definition 2.1 (automaton) *A finite automaton is a tuple $M = (S, s_0, Act, T)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Act is a finite set of actions and $T \subseteq S \times Act \times S$ is a finite set of transitions.*

A transition (s,a,s'), written $s \xrightarrow{a} s'$, represents a transition of the automaton from state s to state s' by executing the action a. An execution of M is a sequence of transitions $(s_0,a_1,s_1)(s_1,a_2,s_2)\ldots$ written $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$, beginning in the initial state s_0.

The next definition introduces the product of automata.

Definition 2.2 (product of automata) *Let $M_1 = (S_1, s_{01}, Act_1, T_1)$ and $M_2 = (S_2, s_{02}, Act_2, T_2)$ be two finite automata, let $-$ be a new symbol which represents an absence of action, and let $f : (Act_1 \cup \{-\}) \times (Act_2 \cup \{-\}) \to Act$ be a partial* synchronization *function, where $f(-,-)$ is undefined.*

The product $M = (S, s_0, Act, T) = M_1 \otimes_f M_2$ is defined as follows:

- $S = S_1 \times S_2$ *is the cartesian product of* S_1 *and* S_2*, with initial state* $s_0 = (s_{01}, s_{02})$*,*
- *the set* T *of transitions contains transition* $(s_1, s_2) \xrightarrow{c} (s'_1, s'_2)$ *iff*
  - $s_1 \xrightarrow{a} s'_1 \in T_1$*,* $s_2 \xrightarrow{b} s'_2 \in T_2$*, and* $c = f(a,b)$
  - *or* $s_1 \xrightarrow{a} s'_1 \in T_1$*,* $s'_2 = s_2$*, and* $c = f(a,-)$
  - *or* $s'_1 = s_1$*,* $s_2 \xrightarrow{b} s'_2 \in T_2$*, and* $c = f(-,b)$

This definition can be easily extended to a set of n automata $M_1, \ldots, M_n$.

LTL is a specification language on infinite behaviors (that can be always obtained by adding self-loops on the deadlock states of a model). Given a set $\mathcal{P}$ of atomic propositions, LTL formulae are defined by the following grammar:

$$\varphi ::= \ p \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where $p \in \mathcal{P}$, $\vee$ is the boolean disjunction and X and U are temporal operators described below. The formulae are interpreted over executions of automata, using a labeling function $\mathcal{L}$ mapping each state to a set of atomic propositions that hold in this state.

Let $M = (S, s_0, Act, T)$ be an automaton and let $\mathcal{L} : S \to 2^{\mathcal{P}}$ be the labeling function. For an execution $e : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$ of M and a LTL formula φ, we note $e, i \vDash \varphi$ when formula φ is satisfied at position i of e. The satisfaction relation is defined inductively by:

	$e, i \vDash p$	*iff* $p \in \mathcal{L}(s_i)$
$\neg$ negation:	$e, i \vDash \neg\varphi$	*iff* $e, i \nvDash \varphi$
$\vee$ disjunction:	$e, i \vDash \varphi_1 \vee \varphi_2$	*iff* $e, i \vDash \varphi_1$ *or* $e, i \vDash \varphi_2$
X next:	$e, i \vDash \mathsf{X}\varphi$	*iff* $e, i+1 \vDash \varphi$
U until:	$e, i \vDash \varphi_1 \mathsf{U} \varphi_2$	*iff* $\exists j \geq i \mid e, j \vDash \varphi_2$ and $\forall i \leq k < j,\ e, k \vDash \varphi_1$

Moreover two temporal operators $\Diamond$ and $\Box$ are defined from *until* by: $\Diamond\varphi = \mathit{true}\mathsf{U}\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. The formula $\Diamond\varphi$ states that φ will be true eventually in the future, and $\Box\varphi$ is satisfied *iff* φ holds forever from now on.

Temporal and boolean operators can be nested. For instance $\Diamond\Box\varphi$ expresses that from some position in the future φ always holds, and $\Box\Diamond\varphi$ states that φ is satisfied infinitely often.

Definition 2.3 *An automaton* M *(with labeling* $\mathcal{L}$*) satisfies* φ *if for each execution* e *of* M*,* $e, 0 \vDash \varphi$*.*

Given an automaton M that represents all possible behaviors of a system and an LTL formula φ describing a requirement on the system, LTL model-checking answers the question whether $M \models \varphi$. When the answer is negative, a counter-example can be exhibited.

For our verification purpose, we opt for two model-checkers: DiVinE [22] and ITS-tools [23]. We choose these model-checkers for their ability to deal with large models and formulae, by using parallel computation for the first one or a symbolic approach for the second one. Moreover they provide several metrics such as the number of states and transitions and they can work on the same input files.

3 Formal model for Mobile Robots Protocols

This section develops the formal modeling we propose for the robots (Section 3.1), the schedulers (Section 3.2), and the system resulting from their composition (Section 3.3). Furthermore, we prove equivalence between the model and its implementation (Section 3.4).

3.1 Robot modeling

The robots execute the same algorithm and have identical behavior [1], hence they can be described by the same automaton. Figure 1 shows a finite automaton modeling a robot's behavior. Recall that robots operate in *Look*, *Compute*, and *Move cycles*.

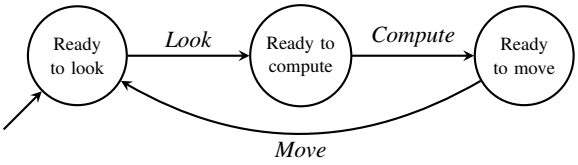


Figure 1: An automaton for the robot behavior

To start a cycle, a robot must take a snapshot of its environment, which is represented by the *Look* transition. Then, it must compute its future location, represented by the *Compute* transition. Finally the robot has to move according to its previous computation, this effective movement is represented by the *Move* transition.

The "Ready to move" state is divided into as many parts as there are possible movements according to the algorithm to be verified.

Note that the original *Look-Compute-Move* model abstracts the precise time constraints (like the computational power or the locomotion speed of robots) and keep only sequences of instantaneous actions, assuming that each robot completes each cycle in finite time. Therefore, the model can be reduced by combining the *Look* and *Compute* phases to obtain the *LC* phase.

3.2 Scheduler modeling

The scheduler organizes robot movements to obtain possible behaviors with respect to FSYNC, SSYNC or ASYNC models. Like the robots, the scheduler is modeled by a finite automaton. For each variant of the execution model, there is one scheduler model. By synchronizing one of these schedulers with robot automata, we obtain an automaton that represents the global behavior of robots in the chosen model. We now describe these scheduler models for a set *Rob* of robots. Unlike robots which have the same behavior regardless of the model, the scheduler is parameterized by the model and the number of robots.

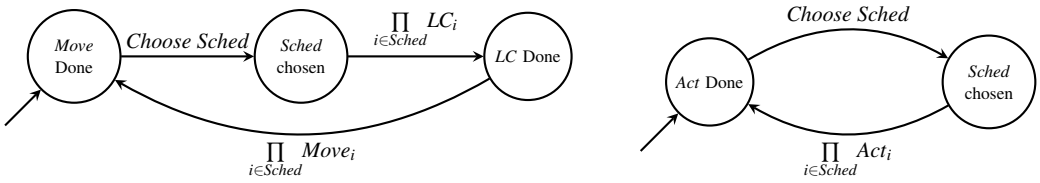
In the sequel we denote by LC_i (respectively $Move_i$), the *LC* (resp. *Move*) phase of i^{th} robot.

And for a subset $Sched \subseteq Rob$, we denote by $\prod_{i \in Sched} LC_i$ (respectively $\prod_{i \in Sched} Move_i$) the synchronized transitions.

The FSYNC model expects that all robots are scheduled for execution at every phase, and operates synchronously. The SSYNC model expects that an arbitrary non-empty subset of robots is scheduled for execution at every phase, and operate synchronously. In the SSYNC case, the automaton consists of a cycle, where a set "*Sched*" is first chosen, then the *LC* and *Move* phases are synchronized for this set. The automaton for SSYNC is described in Figure 2a.

The "*Sched* chosen" state is divided into 2^k states, where k is the number of robots in order to represent all possible sets of $Sched \subseteq Rob$.

In the FSYNC variant, each phase of all robots must be synchronized. In each global cycle, we have $Sched = Rob$, thus all robots are always scheduled and synchronized on every phase. Hence all global cycles are identical.



(a) SSYNC model (b) ASYNC model

Figure 2: The Schedulers automata

The ASYNC model is totally asynchronous. Any finite delay may elapse between *LC* and *Move* phases: A robot can move according to an outdated observation, and any set $Sched \subseteq Rob$ can be scheduled.

The automaton in Figure 2b represents the corresponding scheduler. In each phase a set *Sched* is chosen, and all robots in this set are allowed to act: the action Act_i is either LC_i or $Move_i$ depending on the current state of the i^{th} robot.

3.3 System modeling

A configuration of the system describes the positions of robots on the graph, we denote by *Pos* the set of such positions. In a graph of n nodes with k robots there are $\binom{n}{k}$ possible configurations, thus the total number of states is multiplied by $\binom{n}{k}$. Furthermore, the number of transitions depends on the number of states and on the number of possible movements. Thus to represent the system as an automaton every $Move_i$ transition must be divided according to the shape of the graph.

The model of the system is an automaton $M = (S, s_0, A, T)$ obtained by the synchronized product defined above (section 2), with $A = \prod_{i \in Rob} A_i$, where $A_i = \{LC_i, Move_i, -\}$ for each robot i. From this definition, states are of the form $s = (s_1, \ldots, s_k, c)$ where s_i is the local state of robot i, and

$c : Rob \to Pos$ is the configuration, a mapping associating each robot i with its position $c(i) \in Pos$ in the graph. The initial state is $s_0 = (s_{1_0}, \ldots, s_{k_0}, c_0)$ where s_{i_0} is the initial local state of robot i, and c_0 is the initial configuration.

A transition of the system is labeled by a tuple $a = (a_1, \ldots, a_k)$, where $a_i \in A_i$ for all $1 \leq i \leq k$ and $(s_1, \ldots, s_k, c) \xrightarrow{a} (s'_1, \ldots, s'_k, c')$ *iff* for all i, $s_i \xrightarrow{a_i} s'_i$ and c' is obtained from c by updating the positions of all robots i such that $a_i = Move_i$. To represent the scheduling, we denote by $\prod_{i \in Sched} Act_i$ the action $(a_1, \ldots, a_k)$ such that $a_i = -$ if $i \notin Sched$ and $a_i \in \{LC_i, Move_i\}$ otherwise.

3.4 DVE Implementation

We implement our case study using DVE, the original DiVinE modeling language, which is also interpreted by ITS-tools. A DVE system is composed of processes, that are automata where transitions can be guarded by a "condition" (guard) that determines if the transition can be fired.

In general, algorithms in FSYNC, SSYNC, or ASYNC models are described as a set of guarded actions. Guards are boolean expressions added to the actions of robots to constrain their behavior. An action can be taken only if its guard is evaluated to true. The transcription of these algorithms in DVE is trivial. A guard of the robots algorithm is a guard on a *LC* transition. Transitions have so-called "effects" which are assignments to local or global variables. These correspond to the actions of a guarded-action algorithm. When two transitions can be fired, one of them is chosen nondeterministically.

Although DiVinE language has a tremendous expressive power, we had to deal with an important restriction. The DVE language cannot synchronize more than two automata. Therefore, we implement synchronized actions using a sequential order where look actions (LC_i) are executed first, and the move actions ($Move_i$) afterward.

More formally, we obtain the following system: $M' = (S', S_0, A, T')$ where S' is defined similarly to S, with the addition of a labeling of states (explained below), to indicate if the state is a transient or a steady state. The transition relation is defined as follows. We note $\hat{a_i}$ the tuple of actions where only robot i takes action $a_i \in A_i$: $\hat{a_i} = (-, \ldots, -, a_i, -, \ldots, -)$. By convention, $\hat{a_i} = \varepsilon$ (the empty word) if $a_i = -$, with $s \xrightarrow{\varepsilon} s'$ iff $s' = s$. An action $a = \prod_{i \in Sched} Act_i$ is executed as the sequence of actions $\hat{b_1}, \ldots, \hat{b_k}, \hat{c_1}, \ldots, \hat{c_k}$ where $b_i = LC_i$ if $Act_i = LC_i$ and $-$ otherwise, and similarly, $c_i = Move_i$ if $Act_i = Move_i$ and $-$ otherwise. Hence the single transition $s \xrightarrow{a} s'$ is replaced by a sequence of transitions, where all intermediate states are labeled as transient, while s and s' are steady states.

The following theorem states that our implementation is equivalent to the abstract ASYNC model (see Figure 2b).

Theorem 3.1 *The DiVinE implementation is equivalent to the abstract ASYNC model.*

Proof: Let M be the abstract ASYNC model and M' the model obtained from M as described above. Let $Exec(M)$ and $Exec(M')$ be respectively the set of executions of M and M'. We denote

by $cf(e)$ the sequence of configurations (c component) in $e \in Exec(M)$ and by $cfs(e')$ the sequence of configurations of the steady states in $e' \in Exec(M')$. This notation is extended to the set of executions of M and M' by $cf(Exec(M)) = \{cf(e), e \in M\}$ and $cfs(Exec(M')) = \{cfs(e), e \in M'\}$. We say that two executions $e \in Exec(M)$ and $e' \in Exec(M')$ are equivalent if $cfs(e') = cf(e)$. Since M represents the most "global" behavior and contains all possible executions of the system, we clearly have $cfs(Exec(M')) \subseteq cf(Exec(M))$. To obtain the converse inclusion, we must prove that for each execution $e \in Exec(M)$ we can find an execution $e' \in Exec(M')$ such that e and e' are equivalent.

Let $e \in Exec(M)$. With any transition $t : s \xrightarrow{a} s'$ in e, with $a = (a_1, \ldots, a_k)$, we associate the execution e_t in M' defined above by $s \xrightarrow{\hat{b_1}} s_1 \ldots \xrightarrow{\hat{b_k}} s_k \xrightarrow{\hat{c_1}} s'_1 \ldots \xrightarrow{\hat{c_k}} s'_k$, with all look actions before all move actions. Note that for each i, $\hat{b_i}$ and $\hat{c_i}$ are either ε or belong to $\{\hat{a_1}, \ldots, \hat{a_k}\}$. We now define the execution $e' \in M'$ by replacing all transitions t in e by e_t. We must now prove that e and e' are equivalent.

For this, we show that each transition $t : s \xrightarrow{a} s'$ is equivalent to e_t by examining the ordering of actions. We say that two actions $\hat{a_i}$ and $\hat{a_j}$ commute, written $\hat{a_i} \sim \hat{a_j}$ if for any system state r, if $r \xrightarrow{a_i} r_1 \xrightarrow{a_j} r'$, there exists r'_1 such that $r \xrightarrow{a_j} r'_1 \xrightarrow{a_i} r'$ and vice-versa (the commutativity property is symmetric). This expresses the fact that the state reached is independent of the order of actions $\hat{a_i}$ and $\hat{a_j}$. Clearly, any two *LC* actions commute since they only modify the local state of the robot to which they belong, and only depend on the current configuration which is not updated by LC_i. Similarly, any two *Move* actions on different robots i and j commute, since they successively update the positions of robots i and j in c. Moreover, from the definition of M, all actions $\hat{a_i}$ being simultaneous, the *LC* actions must observe the initial configuration c in the initial steady state s. Therefore, since all *LC* actions appear before the move actions in e_t, this (sequential) execution is equivalent to the (simultaneous) version t. Combining all transitions in e', we obtain that e' and e are equivalent, which concludes the proof.

■

4 Formal verification of Mobile Robots Protocols

Among the protocols designed for discrete settings we choose as case studies the ring exploration with stop and perpetual exclusive ring exploration. For each class of exploration we choose a representative protocol. In both cases we follow the same verification methodology: we first specify the problem in LTL, then translate the protocol in the DiVinE language and verify it. In order to ensure the progress of the protocols, an implicit *fairness* assumption states that all robots must be infinitely often scheduled, which is express in LTL by: $\bigwedge_{i=1}^{k} \Box\Diamond\big(Move_i\big) \wedge \bigwedge_{i=1}^{k} \Box\Diamond\big(LC_i\big)$.

Notation 4.1 *Let G be a ring of n vertexes denoted $V_1, \ldots V_n$. In the sequel we use the following notations: $R[V_j]$ denotes the robot on vertex V_j, and $NbR[V_j]$ the number of robots on vertex V_j.*

4.1 Ring exploration with stop

Flocchini *et al.* first defined [20] the problem of n-ring exploration with stop and proved that the exploration with stop is deterministically impossible when the number of robots k divides n. The authors also proposed a deterministic algorithm to solve exploration with stop using at least 17 robots provided that n and k are co-prime.

Ring exploration with stop specification. For any ring and any initial configuration where robots are located on different vertices, an algorithm solves the exploration with stop problem if within finite time and regardless of the initial placement of the robots, it guarantees the following two properties: *(i)* **exploration:** Each node of the ring is visited by at least one robot, and *(ii)* **ending:** Eventually, the robots must be in a configuration in which they all remain idle (their $Move_i$ action is *idle*). Note that this last property requires robots to "remember" how much of the ring has been explored (*i.e.*, these oblivious robots must be able to distinguish between various stages of the exploration process).

These two properties can be express in LTL as follows: the *exploration* property can be defined by: $\bigwedge_{j=1}^{n} \Diamond\big(nbR[V_j] > 0\big)$. The exploration terminates when all robots remain idle forever which is expressed by the *ending* property: $\bigwedge_{i=1}^{k} \Diamond\Box\big(\neg R_i.Front \wedge \neg R_i.Back\big)$, where $R_i.Front$ (respectively $R_i.Back$) denotes the state of the ith robot when it is ready to move to the Front (resp. to the Back), Front/Back being the possible directions of motion in a ring shaped graph.

Definition 4.1 *An algorithm satisfies the ring exploration with stop specification if from all initial configurations the algorithm verifies: Fairness* $\rightarrow$ $(exploration \wedge ending)$.

Verification Results. The algorithm [20] is composed of three phases: *Set-Up*, *Tower-Creation*, and *Exploration*. In the first phase, from an (arbitrary) initial configuration without tower (a node where more than one robot are simultanesously present), robots gather and occupy a set of consecutive nodes,or two sets of the same size. Once they are gathered the *Tower-Creation* begins: the aim of this phase is to create configurations with tower(s), from which the exploration is feasible. The aim of the last phase is to explore the ring. The formalization of the algorithm is proposed in the appendix. Note that the original paper only presents an informal description of the algorithm.

Our verification proves that the algorithm is correct for all tested instances of k and n that satisfy contraints edicted in the original paper (*i.e* n, k are co-prime and $n,k \geq 17$). We also verify that the algorithm is correct in these settings even for some cases when n and k are not coprime as long as the initial configuration is not periodic (*i.e* there is at most one symmetry axis in the ring). The verification results up to $n = 23$ are presented in the Appendix.

Interestingly, our methodology also permits to refine the correctness bounds of the algorithm for $n,k < 17$ as follows. When k is even the algorithm works as long as $n < k + \lceil k/2 \rceil$ and $k \geq 10$. When k is odd the algorithm works for any $k \geq 5$. Note that the original algorithm provides no rule for these situations.

4.2 Perpetual Ring Exploration

In the sequel we recall the perpetual exclusive ring exploration problem, and propose the verification results for the *Min-Algorithm* [21].

Perpetual Ring Exploration specification. For any ring and any initial configuration where robots are located on different nodes of the ring, an algorithm solves the perpetual exclusive exploration problem if it guarantees these two properties: *(i)* **exclusivity:** No two robots visit the same node or traverse the same edge at the same time, and *(ii)* **liveness:** Each robot visits each node infinitely often. Note that the first property implies that there is never more than one robot on any vertex and that two robots never traverse the same edge at the same time in opposite directions.

The above properties can be express in LTL as follows: the *exclusivity* property is expressed by the conjunction of the *no collision* and the *no switch* properties. *no collision*: $\bigwedge_{j=1}^{n} \Box\Big(NbR[V_j] < 2\Big)$. *no switch*: $\bigwedge_{j=1}^{n} \bigwedge_{i=1}^{k} \bigwedge_{i'=1}^{k} \neg \Diamond\Big(R[V_j] = R_i \wedge R[V_{j+1}] = R_{i'} \wedge R_i.Front \wedge R_{i'}.Back\Big)$.

In order to express that each robot visits all vertices infinitely often, we use the *live* property: $\bigwedge_{j=1}^{n} \bigwedge_{i=1}^{k} \Box\Diamond\big(R[V_j] = R_i\big)$. This property needs the fairness assumption describe above. Hence the *Liveness* property can be expressed by: $Liveness : Fairness \rightarrow live$.

Verification results. The *Min-Algorithm* aims at ensuring that three robots exclusively and perpetually explore any ring of size $n \geq 10$ where n is not a multiple of k. This algorithm is based on a classification of configurations. A closed class of configurations, called legitimate configurations, ensure that after execution of n rounds, all robots have explored the entire ring. When started in a non-legitimate configuration, the protocol ensures convergence towards a legitimate configuration. The algorithm is correct *iff* from any configuration, it converges to a legitimate configuration. The protocol for the legitimate phase (respectively the convergence phase) is provided in the Appendix (as Table 7, resp. Table 8). The results we obtain with this algorithm, for the smallest possible ring of size 10, are presented in Table 1.

nb States	nb Transitions	Memory (kB)	Model	Verification
256 315	737 810	248 668	FSYNC	ok
407 175	881 437	248 840	SSYNC	ok
3 429 715	13 218 742	1 269 432	ASYNC	collision

Table 1: Model-checking of *Min-Algorithm* in the three models for the smallest ring

In order to show factors of state space explosion, we outline the number of states, transitions, the memory used, and the time spend. More importantly, our results show that the algorithm does not satisfy the exclusivity property in the ASYNC model. A counter-example is given by our tools

(and presented as Figure 3 in the Appendix). This counter example allowed the original authors to propose a correct version of their algorithm (that we reproduce in the Appendix). We have verified this new algorithm and Table 2 summarizes the results.

n	nb States	nb Transitions	Memory (kB)	Time
10	1 581 961	6 090 209	1 416 880	6min 45s
11	1 926 385	7 421 315	1 568 748	9min 09s
13	2 716 637	10 476 317	2 252 600	20min 46s
14	3 162 409	12 307 905	2 560 724	26min 54s
16	4 155 385	16 041 365	2 772 188	36min 22s

Table 2: Model-checking of the patched *Min-Algorithm*

5 Conclusion

We demonstrated the feasibility and usefulness of general formal verification through model checking of mobile robot protocols evolving in a discrete space. Our methodology permits not only to find and correct bugs in the protocols (which is especially useful in the more challenging execution models such as ASYNC), but also relieve protocol designers from the burden of manually checking small instances of the problem to be solved, thus permitting them to concentrate on abstract configurations where some global invariants hold. We would like to mention two open issues that are currently under investigation:

1. While our method is parameterized by both k and n, it does not permit to verify whether a protocol is valid for *every* k and n satisfying a particular predicate. Adapting recent advances in parameterized model checking [24] would be a nice way to obtain such results.

2. Our approach aids in the design of mobile robot protocols by permitting to find bugs and loopholes in the overall logic. Going one step further and generating the protocol automatically from the problem would permit to get solutions that are correct by design. We believe that controller synthesis [25] can be extended to obtain such guarantees.

References

[1] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool Publishers, 2012.

[2] A. Clerentin, M. Delafosse, L. Delahoche, B. Marhic, and A. Jolly-Desodt. Uncertainty and imprecision modeling for the mobile robot localization problem. *Auton. Robots*, pages 267–283, 2008.

[3] G. D'Angelo, G. Di Stefano, and A. Navarra. Gathering of six robots on anonymous symmetric rings. In *Structural Information and Communication Complexity*, pages 174–185. Springer Berlin Heidelberg, 2011.

[4] S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In *Structural Information and Communication Complexity*, pages 150–161. Springer Berlin Heidelberg, 2011.

[5] A. Lamani, S. Kamei, F. Ooshita, and S. Tixeuil. Gathering an even number of robots in a symmetric ring without global multiplicity detection. In *Mathematical Foundations of Computer Science*, pages 542,553. Springer Berlin Heidelberg, 2012.

[6] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, pages 1347–1363, 1999.

[7] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, pages 147–168, 2005.

[8] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[9] C. Baier and J. P. Katoen. *Principles of model checking*. MIT press, 2008.

[10] L. Lamport and S. Merz. Specifying and verifying fault-tolerant systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 41–76. Springer Berlin Heidelberg, 1994.

[11] Sandeep S. Kulkarni, Borzoo Bonakdarpour, and Ali Ebnenasir. Mechanical verification of automatic synthesis of fault-tolerant programs. In *Logic Based Program Synthesis and Transformation*, pages 36–52. Springer Berlin Heidelberg, 2004.

[12] R. Guerraoui, T. A. Henzinger, and V. Singh. Model checking transactional memories. *Distributed Computing*, pages 129–145, 2010.

[13] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Algorithmic verification of population protocols. In *Stabilization, Safety, and Security of Distributed Systems*, pages 221–235. Springer Berlin Heidelberg, 2010.

[14] J. Clément, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu. Guidelines for the verification of population protocols. In *Distributed Computing Systems*, pages 215–224. IEEE, 2011.

[15] B. Charron-Bost, H. Debrat, and S. Merz. Formal verification of consensus algorithms tolerating malicious faults. In *Stabilization, Safety, and Security of Distributed Systems*, pages 120–134. Springer Berlin Heidelberg, 2011.

[16] T. Lu, S. Merz, and C. Weidenbach. Towards verification of the pastry protocol using tla$^+$. In *Formal Techniques for Distributed Systems*, pages 244–258. Springer Berlin Heidelberg, 2011.

[17] T. Tsuchiya and A. Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, pages 341–358, 2011.

[18] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. In *Stabilization, Safety, and Security in Distributed Systems*, pages 64–76. Springer Berlin Heidelberg, 2012.

[19] F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols. In *Stabilization, Safety, and Security of Distributed Systems*, pages 282–284. Springer Berlin Heidelberg, 2012.

[20] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *Principles of Distributed Systems*, pages 105–118. Springer Berlin Heidelberg, 2007.

[21] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. *Distributed Computing*, pages 312–327, 2010.

[22] J. Barnat, L. Brim, M. Češka, and P. Ročkai. DiVinE: Parallel Distributed Model Checker (Tool paper). In *Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology*, pages 4–7. IEEE, 2010.

[23] M. Colange, S. Baarir, F. Kordon, and Y. Thierry-Mieg. Towards Distributed Software Model-Checking using Decision Diagrams. In *Computer Aided Verification*, page to appear. Springer Verlag, 2013.

[24] E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In *Computer Aided Verification*, pages 87–98. Springer Berlin Heidelberg, 1996.

[25] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. In *Analysis and Optimization of Systems*. Springer Berlin Heidelberg, 1984.

A Ring exploration with Stop

In the following we propose the formalisation of the algorithm for ring exploration with stop of [20]. This algorithm has been further translated and verified in the DiVinE language. The algorithm allows a set of k identical robots to explore a ring shape graph of n nodes: $u_0, u_1, ..., u_{n-1}$. Nodes are anonymous (*i.e* identical) and the ring is unoriented. Initially there is at most one robot in each node, thus $k < n$. Moreover n and k must be co-prime.

In the following we recall the notations and the definitions used in [20] in order to describe the algorithm. It should be noted that in the original version the algorithm is proposed in a verbose mode. In order to translate the algorithm in the DiVinE language we did a pre-processing phase that consists in expressing the rules of the algorithm in terms of guarded actions.

Let $d_i(t)$ denote the multiplicity at node i at time t. It permits to detect the presence of towers (multiple robots on one node). Let $\delta^{+j}(t)$ denote the sequence $\delta^{+j}(t) :< d_j(t)d_{j+1}(t)...d_{j+n-1}(t) >$ and let $\delta^{-j}(t)$ denote the sequence $\delta^{-j}(t) :< d_j(t)d_{j-1}(t)...d_{j-(n-1)}(t) >$. The unordered pair of sequences $\delta^{+j}(t)$ and $\delta^{-j}(t)$ describes the configuration of the system at time t viewed from node u_j. Let $\Delta^+(t) = \delta^{+j}(t) : 0 \leq j < n$ and $\Delta^-(t) = \delta^{-j}(t) : 0 \leq j < n$. We will denote by $\delta_{max}(t)$ the lexicographically maximum sequence in $\Delta^+(t) \cup \Delta^-(t)$. It follows [20] that there is at most one maximal sequence in each of $\Delta^+(t)$ and $\Delta^-(t)$. A configuration is said to be symmetric if the maximal sequences in $\Delta^+(t)$ and $\Delta^-(t)$ are equal, and asymmetric otherwise. Let $\delta^{max\_r_i}(t)$ be the maximal observation for the i^{th} robot, and $\delta^{min\_r_i}(t)$ be his minimal observation.

Definition A.1 (interdistance) *Let **d** be the minimum distance between all pairs of distinct robots in the configuration. **d** is called the **interdistance**.*

Definition A.2 (neighbor) *Two robots are **neighbors** if in a least one direction there is no robot between them.*

Definition A.3 (block) *A **block** is a maximal set of robots (at least 2) , forming a line, where neighbors are separated by **d** free nodes.*

Definition A.4 (border robot) *A robot r is the **border** of a block if it is one of the extremal robot of this block.*

Definition A.5 (leading block) *A block b is said **leading** if it is a block for which one border is a robot whose view is maximal.*

Definition A.6 (isolated robot) *A robot is **isolated** if he is not part of a block.*

The algorithm is divided in three phases, the Set-Up phase, the Tower-Creation phase and the Exploration phase. We will formally describe each phase of the algorithm as it can be done by each robot. The first phase is the Set-Up phase. It is described in subsection A.1, it permits to gather all robots in one group or two groups of the same size, this last configuration is called *no-towers-final*.

In the second phase, the goal is to create one or two towers per block according to the arity of the blocks. This phase is described in subsection A.2. The last phase permits the exploration of the ring, and is described in subsection A.3.

Notation A.1 *In the following we will denote by B the set of blocks, R the set of robots, and T the set of towers. For our purpose we define r_i the current robot. Moreover the algorithm uses the predicates: neighbor(r_1, r_2) to express that robots r_1,r_2 are neighbors, border(r,b) to express that the robot b is a border of the block b, leading(b) in order to express that b is a leading block, and isolated(r) to express that r is an isolated robot.*

Robots have no chirality, thus we describe the *Compute* transition according to the current view of the robots. Each view is described by δ^+ and δ^-, the compute movement can be described similarily as the view by μ^+ or μ^-. When a robot executes a μ^+(respectively μ^-) it moves in the same direction as the δ^+(resp.δ^-) view. We define also the $\mu^=$ movement as a movement which can be either a μ^+ or a μ^- movement.

A.1 The Set-Up Phase

The Set-Up phase is the first phase of the algorithm. It is assumed that all initial configurations do not contain any tower. There are four types of configurations that form a partition of all possible configurations without tower. Type B, C, D when there is no isolated robot, and A otherwise. Configurations of type D are the Set-Up final configurations. Configurations of type C are configurations that contain one block or two blocks of the same size. Configurations of type B are all the remaining configurations, where there is no isolated robot.

We describe the protocol that robots execute for each type of configuration. All configurations in this phase share the following predicate:

- Set-Up(): $|T| = 0$

A.1.1 Type A Configurations

Notation dans Type A:

- S = $\max_{\text{size}_b}(\text{Neighbor}(r,b) \wedge \text{Isolated}(r))$
- Move$(r,b) = \mu^-$ if $dist^+(r,b) > dist^-(r,b)$ and μ^+ otherwise

Predicates in Type A:

- Type A() : Set-Up() $\wedge d \geq 0 \wedge \exists r \in R, Isolated(r)$
- Close-to-S(r,b) : $Isolated(r) \wedge Neighbor(r,b) \wedge b.size = S$

Sets in Type A:

- Closest-to-S = $\{r, \min_{dist(r,b)}(\text{close-to-S}(r,b))\}$

Action in Type A:

A_A: TypeA() $\wedge \exists b,\ r_i \in$ closest-to-S $\rightarrow$ Move(r_i, b)

A.1.2 Type C and D Configurations

Notation in Type C D :

- Move(r) : μ^- if $\delta^{+r_i} > \delta^{-r_i}$ and μ^+ otherwise

predicates in Type C and D Configurations :

- Type CD() : Set-Up() $\wedge \forall b \in B$, $Leading(b)$
- Type C() : $d \geq 2 \wedge$ Type CD()
- Type D() : $d = 1 \wedge$ Type CD()

Action in Type C configurations:

A_C: TypeC()$\wedge$ Leading$(r_i) \rightarrow$ Move(r_i)

Action in Type D Configurations:

A_D: TypeD() $\rightarrow$ exploration()

A.1.3 Type B Configurations

Notations in Type B:

- $\delta^{\max}(r)$: max$(\delta^{+r}(t), \delta^{-r}(t))$

Predicates in Type B:

- Type B() : Set-Up()$\wedge\ \nexists r \in R$, Isolated$(r)\wedge\ d \geq 1 \wedge \exists b$, $\neg$Leading(b)

A.1.4 Type B2 Configurations

Notations in Type B2 :

- s: $\min_{b.\text{size}}(b)$
- S: : $\max_{b_1.\text{size}}$ (Neighbor$(b_1, b_2) \wedge b_2$.size = s)
- dist: $\min_{distance(b_1,b_2)}$(Neighbor$(b_1, b_2) \wedge b_2$.size = s$\wedge b_1$.size = S)
- Largest-view($\mathcal{T}$): $\max_{\delta^{\max}}(r \in \mathcal{T})$
- Move(r,b) : μ^- if $dist^+(r,b) > dist^-(r,b)$ and μ^+ otherwise

Sets in Type B2 Configurations :

- $\mathcal{T}$: $\{r$, border$(r, b_1) \wedge b_1$.size = s $\wedge$ Neighbor$(r, b_2) \wedge b_2$.size = S $\wedge$ distance$(b_1, b_2) = dist\}$

Predicates in Type B2 Configurations:

- Type B2(): Type B() $\wedge \exists b_1, b_2 \in B$, $b_1.size \geq b_2.size$

Actions in Type B2 Configurations:

A_{B2}: Type B2$() \wedge r_i \in \mathcal{T} \wedge \delta^{\max}(r_i) =$Largest-view$(\mathcal{T}) \wedge \exists b$, (Neighbor$(r_i,b) \wedge b_2$.size = S$\wedge$distance$(r_i,b) = dist) \rightarrow$ Move(r_i,b)

A.1.5 Type B1 Configurations

Notations in Type B1 Configurations:

- Between$(b_1,b_2) = (x,y)$ A pair of integers x, y where x and y are the number of blocks between b_1 and b_2 in the two directions.
- Move_Outside(r,b) : μ^- if $\delta^{+r} > \delta^{-r}$ and μ^+ otherwise

Sets in Type B1 Configurations:

- $\mathcal{L}$: $\{r,$Leader$(r)\}$
- SymR: $\{(r_1,r_2)\}$, symmetric$(r_1,r_2) \wedge \exists b \in B$ border(r_1,b).
  The set of pairs of symmetric robots among the robots at the border of a block.
- SymB: $\{(b_1,b_2)\}$, symmetric$(b_1,b_2) \wedge$ Between$(b_1,b_2) \geq (3,3)$
  The set of pairs of symmetric blocks separated by at least three blocks in each side.
- Closest-Pairs(Set-of-Pairs): $\{(r_1,r_2)\}$, distance$(r_1,r_2) =$ min$_{\text{distance}}$(Set-of-Pairs)$\wedge \neg$Neighbor(r_1,r_2).
  The set of closest pair among the pair of Set-of-pairs such that this robots are not neighbors.
- Smallest-view(Block-Set): $\{r_1\}$ such that $\forall b \in$ Block-Set $\exists r_1,r_2$, Border$(r_1,b) \wedge$ Border$(r_2,b) \wedge$ $\delta_{min_{r_1}}(t) < \delta^{min_{r_2}}(t)$

Predicates in Type B1 Configurations:

- Type B1(): Type B$() \wedge \forall b_1,b_2 \in B b_1$.size =$b_2$.size
- symmetricRobots(r_1,r_2): $\delta_{\max r_1}(t) = \delta^{\max_{r_2}}(t)$ and $r_1, \neq r_2$
- symmetricBlocks(b_1,b_2): $\exists r_1,r_2$, border$(r_1,b_1) \wedge$border$(r_2,b_2) \wedge$ symmetric$(r_1,r_2) \wedge b_1$.size = b_2.size

Actions in Type B1 Configurations:

A_{B11}: Type B1$() \wedge |\mathcal{L}| = 1 \wedge$ Learder$(r) \wedge \exists b$, Border$(r,b) \rightarrow$Move_Outside(r,b)
A_{B121}: Type B1$() \wedge |\mathcal{L}| = 2 \wedge b$.size =$2 \wedge \exists b$border$(r,b) \wedge r \in$ Smallest-viewSymB $\rightarrow$Move_Outside(r,b)
A_{B122}: Type B1$() \wedge |\mathcal{L}| = 2 \wedge b$.size $\neq 2 \wedge r_i \in$ Closest-Pairs(SymR)$\wedge$border$(r_i,b) \rightarrow$Move_Outside(r_i,b)

A.2 The Tower-Creation Phase

The Set-Up phase final configurations are of the form: one block of odd size, or one block of even size, or two blocks of odd size, or two blocks of even size. From these configurations towers are made during the TowerCreation phase. The rules executed in this phase are of he form `view` $\xrightarrow{p}$ `view`, where `view` represents the view of the current robot, and p the number of robots that move

synchronously. Moreover a view is a sequence of F, R, where F_y represents y free consecutive nodes and R_x represents x consecutive nodes occupied by one robot. A tower is represented by the symbol T.

Tower-Creation Phase:				
$R1_0$::	(R_a, F_x, R_a)	$\rightarrow$	(R_{a-1}, F_x, R_a, F_1)	K odd and $2a+1=K$
$R2_0$::	(R_{a-1}, F_x, R_a)	$\xrightarrow{1}$	$(R_{a-2}, F_x, R_{a-2}, T, F_2)$	K even and $2a=K$
		$\xrightarrow{2}$	$(R_{a-2}F_x, R_{a-2}, T, F_2)$	
$R2_1$::	$(R_{a-1}, F_x, R_{a-2}, T, F_1)$	$\rightarrow$	$(R_{a-2}, Fx, R_{a-2}, T, F_2)$	K even and $2a=K$
$R3_0$::	$(R_a, F_x, R_{K/2}, F_y, R_a)$	$\xrightarrow{1}$	$(R_{a-1}, F_x, R_{K/2}, F_y, R_a, F_1)$	K even, $2a+1=K$ and $y<x$
		$\xrightarrow{2}$	$(R_a-1, F_x, R_{a-1}, T, F_1, R_a, F_y, R_a, F_1)$	
$R3_1$::	$(R_a, F_x, R_{a-1}, T, F_1, R_a, F_y, R_a)$	$\rightarrow$	$(R_{a-1}, F_x, R_{a-1}, T, F_1, R_a, F_y, R_a, F_1)$	
$R4_0$::	$(R_a, F_x, R_{K/2}, F_y, R_{a+1})$	$\xrightarrow{1}$	$(R_{a-1}, F_x, R_{k/2}, F_y, R_{a+1}, F_1)$	K even and $k/2=2a+2$
		$\xrightarrow{2}$	$2_1 : (R_{a-1}, F_x, R_{k/2}, F_y, R_{a-1}, T, F_2)$	
		$\xrightarrow{2}$	$2_2 : (R_{a-1}, F_x, R_{a-1}, T, F_1, R_{a+1}, F_y, R_{a+1}, F_1)$	
		$\xrightarrow{2}$	$2_3 : (R_{a-1}, F_x, R_{a+1}, F_1, T, R_{a-1}, F_y, R_{a+1}, F_1)$	
		$\xrightarrow{3}$	$3_1 : (R_{a-1}, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, Ra+1, F_1)$	
		$\xrightarrow{3}$	$3_2 : (R_{a-1}, F_x, R_{a+1}, F_1, T, R_{a-1}, F_y, Ra-1, T, F_2)$	
		$\xrightarrow{3}$	$3_3 : (R_{a-1}, F_x, R_{a-1}, T, F_1, R_{a+1}, F_y, Ra-1, T, F_2)$	
		$\xrightarrow{4}$	$4_0 : (R_{a-1}, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, R_{a-1}, T, F_2)$	
$R4_{11}$::	$(R_a, F_x, R_{K/2}, F_y, R_{a-1}, T, F_1)$	$\xrightarrow{1}$	(2_1)	
		$\xrightarrow{2}$	$(3_2$ or $3_3)$	
		$\xrightarrow{3}$	$(R_{a-1}, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, R_{a-1}, T, F_2))$	
$R4_{12}$::	$(R_a, F_x, R_{a-1}, T, F_1, R_{a-1}, F_y, R_{a+1})$	$\xrightarrow{1}$	(2_2)	
		$\xrightarrow{2}$	$(3_1$ or $3_3)$	
		$\xrightarrow{3}$	$(R_{a-1}, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, R_{a-1}, T, F_2))$	
$R4_{13}$::	$(R_a, F_x, R_{a+1}, F_1, T, R_{a-1}, F_y, R_{a+1})$	$\xrightarrow{1}$	(2_3)	
		$\xrightarrow{2}$	$(3_2$ or $3_1)$	
		$\xrightarrow{3}$	(4_0)	
$R4_{21}$::	$(R_a, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, R_{a+1})$	$\xrightarrow{1}$	(3_1)	
		$\xrightarrow{2}$	(4_0)	
$R4_{22}$::	$(R_a, F_x, R_{a+1}, F_1, T, R_{a-1}, F_y, R_{a-1}, T, F_1)$	$\xrightarrow{1}$	(3_2)	
		$\xrightarrow{2}$	(4_0)	
$R4_{21}$::	$(R_a, F_x, R_{a-1}, T, F_1, R_{a+1}, F_y, R_{a-1}, T, F_1)$	$\xrightarrow{1}$	(3_3)	
		$\xrightarrow{2}$	(4_0)	
$R4_3$::	$(R_a, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, R_{a-1}T, F_1)$	$\rightarrow$	(4_0)	

Table 3: Rules of the Tower Creation phase

A.3 The Exploration Phase

The exploration phase is the last phase of the algorithm. It starts when all towers of the preceding phase are created, and is described by the two tables below.

Exploration Phase:		
$E1_0$::	$(F_x, R_a, F_1, T, R_{a-2}, F_y) \rightarrow$ $(F_{x-1}, R_a, F_1, T, R_{a-2}, F_y+1)$	K odd, $2a+1=K$ x ≥ 1
$E2_0$::	$(F_x, R_1, F_y, R_{a-3}, T, F_2, T, R_{a-3}, F_z) \xrightarrow{1}$ $(F_{x-1}, R_1, F_y, R_{a-3}, T, F_2, T, R_{a-3}, F_{z+1}) \xrightarrow{2}$ $(F_{x-2}, R_1, F_{y+1}, R_{a-3}, T, F_2, T, R_{a-3}, F_{z+1})$	K even, $2a=K$ $x>0$, $z<(N-K+2)/2$ $x>0$, $z<(x+y+z)/2$ and $y<(x+y+z)/2$
$E3_{01}$::	$(F_g, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_c, R_1, F_d, R_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_f) \xrightarrow{1}$ $(F_{g-1}, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_c, R_1, F_d, R_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{2}$ $(F_{g-2}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_c, R_1, F_d, R_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{2}$ $(F_{g-1}, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}, R_1, F_{d-1}, R_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{2}$ $(F_{g-1}, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_c, R_1, F_{d-1}, R_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{3}$ $(F_{g-2}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}, R_1, F_{d-1}, R_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{3}$ $(F_{g-2}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_c, R_1, F_{d-1}, R_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{3}$ $(F_{g-1}, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}, R_1, F_{d-2}, R_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}) \xrightarrow{4}$ $4_0 : (F_{g-2}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}, R_1, F_{d-2}, R_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_{f+1})$	K even, $2a+1=K/2$and $d\geq 0 \wedge g \geq 0$ $f<(f+g+b)/2$ $f<(f+g+b)/2 \wedge b<(f+g+b)/2$ $(f<(f+g+b)/2) \wedge (e<(e+d+c)/2)$ $(f<(f+g+b)/2) \wedge (c<(e+d+c)/2)$ $(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (c<(e+d+c)/2)$ $(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (e<(e+d+c)/2)$ $(e<(e+d+c)/2 \wedge c<(e+d+c)/2) \wedge (f<(f+g+b)/2)$ $(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (e<(e+d+c)/2 \wedge c<(e+d+c)/2))$
$E3_{02}$::	$(F_d, B_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_f, R_1, F_g, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_c) \xrightarrow{1}$ $(F_{d-1}, B_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_f, R_1, F_g, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{2}$ $(F_{d-2}, B_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_f, R_1, F_g, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{2}$ $(F_{d-1}, B_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_f, R_1, F_{g-1}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{2}$ $(F_{d-1}, B_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}, R_1, F_{g-1}, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{3}$ $(F_{d-2}, B_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}, R_1, F_{g-1}, R_1, F_b, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{3}$ $(F_{d-2}, B_1, F_{e+1}, R_{a-1}, F_1, T, R_{a-2}, F_f, R_1, F_{g-1}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{3}$ $(F_{d-1}, B_1, F_e, R_{a-1}, F_1, T, R_{a-2}, F_{f+1}, R_1, F_{g-2}, R_1, F_{b+1}, R_{a-2}, T, F_1, B_{a-1}, F_{c+1}) \xrightarrow{4} (4_0)$	K even, $2a+1=K/2$ and $d\geq 0 \wedge g \geq 0$ $c<(c+d+e)/2$ $e<(e+d+c)/2 \wedge c<(e+d+c)/2$ $(b<(f+g+b)/2) \wedge (c<(e+d+c)/2)$ $(f<(f+g+b)/2) \wedge (c<(e+d+c)/2)$ $(e<(e+d+c)/2 \wedge c<(e+d+c)/2) \wedge (f<(f+g+b)/2)$ $(e<(e+d+c)/2 \wedge c<(e+d+c)/2) \wedge (b<(f+g+b)/2)$ $(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (c<(e+d+c)/2)$ $(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (e<(e+d+c)/2 \wedge c<(e+d+c)/2))$

Table 4: First rules of the Exploration phase

Exploration Phase: Continuation		
$E3_{03}$::	$(F_g,R_1,F_b,R_{a-2},T,F_1,B_{a-1},F_c,T,F_e,R_{a-1},F_1,T,R_{a-2},F_f) \xrightarrow{1}$	K even, $2a+1=K/2$ and $g\geq 0$
	$(F_{g-1},R_1,F_b,R_{a-2},T,F_1,B_{a-1},F_c,T,F_e,R_{a-1},F_1,T,R_{a-2},F_{f+1}) \xrightarrow{2}$	$f<(f+g+b)/2$
	$(F_{g-2},R_1,F_{b+1},R_{a-2},T,F_1,B_{a-1},F_c,T,F_e,R_{a-1},F_1,T,R_{a-2},F_{f+1})$	$f<(f+g+b)/2 \wedge b<(f+g+b)/2$
$E3_{04}$::	$(F_d,B_1,F_e,R_{a-1},F_1,T,R_{a-2},F_f,T,F_b,R_{a-2},T,F_1,B_{a-1},F_c) \xrightarrow{1}$	K even, $2a+1=K/2$
	$(F_{d-1},B_1,F_e,R_{a-1},F_1,T,R_{a-2},F_f,T,F_b,R_{a-2},T,F_1,B_{a-1},F_{c+1}) \xrightarrow{2}$	$c<(c+d+e)/2$
	$(F_{d-2},B_1,F_{e+1},R_{a-1},F_1,T,R_{a-2},F_f,T,F_b,R_{a-2},T,F_1,B_{a-1},F_{c+1})$	$e<(e+d+c)/2 \wedge c<(e+d+c)/2$
$E4_{01}$::	$(F_g,R_1,F_b,R_{a-2},T,F_1,B_{a-1},F_c,R_1,F_d,R_1,F_e,R_{a-1},F_1,T,R_{a-2},F_f) \xrightarrow{1}$	K even, $2a+2=K/2$
	$(F_{g-1},R_1,F_b,R_{a-2},T,F_2,T,B_{a-2},F_c,R_1,F_d,R_1,F_e,R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{2}$	$f<(f+g+b)/2$
	$(F_{g-2},R_1,F_{b+1},R_{a-2},T,F_2,T,B_{a-2},F_c,R_1,F_d,R_1,F_e,R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{2}$	$f<(f+g+b)/2 \wedge b<(f+g+b)/2$
	$(F_{g-1},R_1,F_b,R_{a-2},T,F_2,T,B_{a-2},F_{c+1},R_1,F_{d-1},R_1,F_e,R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{2}$	$(f<(f+g+b)/2) \wedge (e<(e+d+c)/2)$
	$(F_{g-1},R_1,F_b,R_{a-2},T,F_2,T,B_{a-2},F_c,R_1,F_{d-1},R_1,F_{e+1},R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{3}$	$(f<(f+g+b)/2) \wedge (c<(e+d+c)/2)$
	$(F_{g-2},R_1,F_{b+1},R_{a-2},T,F_2,T,B_{a-2},F_{c+1},R_1,F_{d-1},R_1,F_e,R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{3}$	$(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (c<(e+d+c)/2)$
	$(F_{g-2},R_1,F_{b+1},R_{a-2},T,F_2,TB_{a-2},F_c,R_1,F_{d-1},R_1,F_{e+1},R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{3}$	$(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (e<(e+d+c)/2)$
	$(F_{g-1},R_1,F_b,R_{a-2},T,F_2,T,B_{a-2},F_{c+1},R_1,F_{d-2},R_1,F_{e+1},R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{4}$	$(e<(e+d+c)/2 \wedge c<(e+d+c)/2) \wedge (f<(f+g+b)/2)$
	$(F_{g-2},R_1,F_{b+1},R_{a-2},T,F_2,T,B_{a-2},F_{c+1},R_1,F_{d-2},R_1,F_{e+1},R_{a-2},T,F_2,T,R_{a-2},F_{f+1})$	$(f<(f+g+b)/2 \wedge b<(f+g+b)/2) \wedge (e<(e+d+c)/2 \wedge c<(e+d+c)/2))$
$E4_{02}$::	$(F_g,R_1,F_b,R_{a-2},T,F_2,T,B_{a-1},F_c,T,F_e,R_{a-1},T,F_2,T,R_{a-2},F_f) \xrightarrow{1}$	K even, $2a+2=K/2$ and $g\geq 0$
	$(F_{g-1},R_1,F_b,R_{a-2},T,F_2,T,B_{a-2},F_c,T,F_e,R_{a-2},T,F_2,T,R_{a-2},F_{f+1}) \xrightarrow{2}$	$f<(f+g+b)/2$
	$(F_{g-2},R_1,F_{b+1},R_{a-2},T,F_2,T,B_{a-2},F_c,T,F_e,R_{a-2},T,F_2,T,R_{a-2},F_{f+1})$	$f<(f+g+b)/2 \wedge b<(f+g+b)/2$

Table 5: Second rules of the Exploration phase

B Model-checking of the ring exploration with stop

k	n	nb States	nb Transitions	Memory (kB)
5	6	6	18	163 600
5	7	500	1 410	171 084
5	8	2 786	10 596	183 840
5	9	5 533	18 746	207 788
5	10	5 123 204	25 755 007	668 368
5	11	7 827	23 898	299 980
5	12	13 996	61 822	380 244
5	13	17 149	82 902	491 708
5	14	30 680	157 829	637 840
5	15	19 784 312	130 057 237	2 667 812
5	16	12 418	73 688	1 081 736
5	17	33 004	207 642	1 401 280
5	18	10165	66 120	1 790 644
7	8	8	24	171 396
7	9	2 764	7 576	201 096
7	10	3 022	9 220	270 676
7	11	16 471	56 390	437 876
7	12	18 347	42 448	754 680
7	13	20 272	83 706	1 352 120
10	11	11	33	190 884
10	12	1 834	4 868	460 716
10	13	7 924	23 731	756 000
10	14	8 357	27 524	2 135 944
k	**n**	**nb States**	**Time**	**Memory (kB)**
17	18	4	0:0:04	60 984
18	19	20	0:0:04	66 256
17	19	426	0:25:29	1 622 180
19	20	4	0:0:8	88 168
18	20	248	0:12:10	2 130 824
17	20	849	8:8:00	22 045 016
20	21	20	0:0:08	100 136
19	21	533	1:8:12	3 632 488
18	21	630	3:0:52	9 427 620
17	21	1 663	18:40:07	55 287 000
21	22	4	0:0:12	123 920
20	22	533	1:58:27	5 913 880
19	22	1040	8:25:22	30 243 392
18	22	1380	20:32:45	100 327 682

Table 6: Model-checking of the ring exploration with stop

C Perpetual Ring Exploration

In the following we detail the *Min-Algorithm* from [21] which ensures that three robots always exclusively and perpetually explore any ring of size $n \geq 10$ where n is not a multiple of k. This algorithm is based on a classification of configurations:

Definition C.1 *For k robots in the n-node ring, a configuration is a circular and non oriented alternating sequence of symbols R and F, indexed by integers: R_i stands for i consecutive nodes, each of them occupied by a robot, and F_j stands for j consecutive nodes free of robots.*

A closed class of configurations is outlined. These configurations, called legitimate configurations, are defined by: $C0 = (R_2, F_2, R_1, F_z)$, $C1 = (R_1, F_1, R_1, F_2, R_1, F_z)$ and $C2 = (R_2, F_3, R_1, F_z)$ with $z \in \{0, 1, 2, 3\}$. The phase occurring on these configurations is called the *Legitimate* phase. When started in a legitimate configuration the protocol always moves into a legitimate configuration, after the execution of n rounds, all robots have explored the ring. When started in a non-legitimate configuration the protocol ensures the convergence towards a legitimate configuration thanks to the convergence phase. The algorithm is correct *iff* from any configuration, it converges to a legitimate configuration. The legitimate phase (respectively the convergence phase) can be seen in Table 7 (resp. Table 8)

Legitimate Phase: $z \neq \{0,1,2,3,4\}$			
$RL1$::	(R_2, F_2, R_1, F_z)	$\rightarrow$	$(R_1, F_1, R_1, F_2, R_1, F_{z-1})$
$RL2$::	$(R_1, F_1, R_1, F_2, R_1, F_z)$	$\rightarrow$	(R_2, F_3, R_1, F_z)
$RL3$::	(R_2, F_3, R_1, F_z)	$\rightarrow$	(R_2, F_2, R_1, F_{z+1})

Table 7: Rules of *Min-Algorithm* legitimate phase

Convergence Phase: Execution starting from special configurations.				
$RC1$::	(R_2, F_y, R_1, F_z)	$\rightarrow$	$(R_2, F_{min(y,z)}, R_1, F_{max(y,z)+1})$	with $y \neq z \neq \{1,2,3\}$
$RC2$::	$(R_1, F_x, R_1, F_y, R_1, F_y)$	$\rightarrow$	$(R_1, F_x, R_1, F_{y-1}, R_1, F_{y+1})$	with $x \neq y \neq 0$
$RC3$::	$(R_1, F_x, R_1, F_y, R_1, F_z)$	$\rightarrow$	$(R_1, F_{x-1}, R_1, F_{y+1}, R_1, F_z)$	with $x < y < z$
$RC4$::	(R_3, F_z)	$\rightarrow$	(R_2, F_1, R_1, F_{z-1})	when 1 robot executes
		$\rightarrow$	$(R_1, F_1, R_1, F_1, R_1, F_{z-2})$	when 2 robots execute
$RC5$::	(R_2, F_1, R_1, F_z)	$\rightarrow$	(R_2, F_2, R_1, F_{z-1})	

Table 8: Rules of *Min-Algorithm* convergence phase

Model-checking does not only allow to know whether a system satisfies some given properties, it also gives a counter-example when the system fails to satisfy these properties. A counter-example is an execution that does not satisfy the properties. The counter example given for the ASYNC model is shown in Figure 3. Every ring represents a configuration, a change of configuration

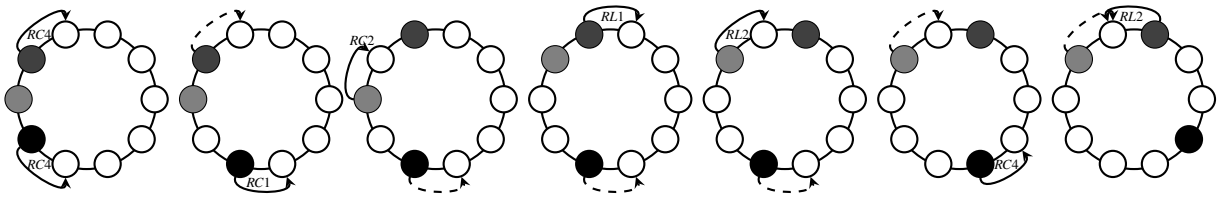


Figure 3: Counter-example

occurs when a robot moves. In each configuration a computation is represented by a full arrow, and a computation made from an outdated snapshot by a dotted arrow.

Thus a new algorithm is proposed by the authors of [21]. The legitimate phase is the same, only the convergent phase changes, more precisely, only rule *RC*5 changes to avoid collisions which arose from the previous rules, when movements computed on obsolete observations are taken into account. The new *RC*5 rule is:

$$RC5 \;::\; (R_2, F_1, R_1, F_z) \;\rightarrow\; (R_1, F_1, R_1, F_1, R_1, F_{z-1})$$